Automated Disk Drive Characterization

Jiri Schindler Gregory R. Ganger

December 1999

CMU-CS-99-176

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

DIXtrac is a program that automatically characterizes the performance of modern disk drives. This report describes and validates DIXtrac's algorithms, which extract accurate values for over 100 performance-critical parameters in 2 – 6 minutes without human intervention or special hardware support. The extracted data include detailed layout and geometry information, mechanical timings, cache management policies, and command processing overheads. DIXtrac is validated by configuring a detailed disk simulator with its extracted parameters; in most cases, the resulting accuracies match those of the most accurate disk simulators reported in the literature. To date, DIXtrac has been successfully used on over 20 disk drives, including eight different models from four different manufacturers.

Jiri Schindler was supported by funding from Data Storage Systems Center and Parallel Data Laboratory at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the respective centers or Carnegie Mellon University.

Keywords: Disk Performance Characterization, Disk Modeling, File Systems.

1 Introduction

One approach to bridging the performance gap between permanent and volatile storage is to fully exploit the current level of technology in the most prevalent permanent storage devices, i.e. the hard disk drives. With detailed knowledge of the underlying technology and its limits, one can achieve the maximal possible performance while not sacrificing generality. Therefore, it is advantageous to obtain as much knowledge about disk drives as possible and to utilize it to aggressively tune performance.

While detailed models of disk behavior are highly desirable for aggressive scheduling algorithms [14, 13, 17, 1] and comprehensive system modeling [9], obtaining accurate data from state-of-the-art disk drives for detailed disk models (e.g., [10, 3, 5]) is at best tedious. Although methodologies for accurate disk models are well understood [10] and general techniques for on-line extraction of parameters from SCSI disk drives have been proposed [18, 1], there is no available system for easily retrieving parameters from a given disk drive.

This paper presents DIXtrac (DIsk eXtraction), a program that can quickly and automatically characterize disk drives that understand Small Computer System Interface (SCSI) [2]. Without human intervention, DIXtrac can discover accurate values for over 100 performance-critical parameters. It runs on standard Linux 2.2 systems, and requires no special hardware or operating system support. By automating this process, DIXtrac greatly simplifies the process of collecting disk drive characterizations.

Automatic extraction of parameters from disks poses several challenges compared to manual, or user-assisted, approaches. If the extraction is to be general enough to work for disk drives of different vendors, it must use methods that will work for all drives and dynamically adapt when extracting from disks that support only a subset of the the interface features. DIXtrac's extraction algorithms use only widely-supported interface features and include multiple approaches for discovering many characteristics. As with any expert system, DIXtrac can only extract parameters and understand behavior that it knows about ahead of time. DIXtrac's algorithms have now been successfully tested on 8 disk models from 4 vendors.

Validation experiments show that DIXtrac can quickly and accurately characterize a variety of disk drives. Most characterizations take less than 3 minutes to complete, with the longest extraction taking 6 minutes. By configuring a detailed disk simulator with the extracted values, their accuracy was empirically validated. In most cases, the DIXtrac-configured simulator is comparable to the

most accurate disk simulations reported in the literature.

The remainder of the report is organized as follows. Section 2 discusses related work. Section 3 overviews disk drive characterization. Section 4 describes DIXtrac's extraction algorithms. Section 5 details DIXtrac's implementation on Linux. Section 6 quantifies the extraction time and accuracy of DIXtrac's characterization of several disk drives.

2 Related Work

Ruemmler & Wilkes [10] present a strong case for detailed disk drive simulators and their use in computer systems research. They identify performance-critical parameters of a disk drive and compare behavior of a real disk drive to a disk simulator that progressively adds more performance-critical parameters. As their simulator gets more detailed and takes into account more disk parameters, its performance more closely approximates the performance of the real disk drive. However, they do not discuss how they obtained parameters from real disk drives that were used in their models. The observations made by Ruemmler & Wilkes have been validated by Kotz et al. [5] who built a detailed simulator of the HP 97560 disk drive.

Ganger et al. have made available a highly-configurable disk simulator called DiskSim [3]. The simulator models several independent parts of the disk drive system: device drivers, busses, controllers, adapters, and disk drives. DiskSim offers over 100 parameters for the disk drive module, though some are dependent on others or meaningful only in certain cases. DIXtrac complements detailed simulators like DiskSim in that it automatically extracts the necessary parameters from a SCSI disk drive, allowing them to be fed into DiskSim for later system simulation and/or testing.

Talagala et al. extract disk geometries, mechanical overheads and layout parameters using microbenchmarks consisting of only read and write requests [16]. By timing requests with progressively increasing request strides, they determine the various parameters. Unlinke DIXtrac, their approach is independent of the disk's interface and thus works for potentially any disk. However, as a consequence, their approach is only able to determine a small subset of parameters and does so with less precission than DIXtrac.

Via a combination of interrogative and empirical extraction techniques, Worthington et al. describe how to retrieve parameters for several disk drives [18]. Interrogative extraction uses the wealth of SCSI command options to determine information, or just hints, about geometry, format, and caches. Empirical techniques measure response times for sequences of read and write requests.

Using different sequences, various performance critical parameters can be extracted. With these techniques, they were able to extract parameters for a set of disk drives.

Using similar techniques, Aboutabl et al. describe the extraction of many of the same disk parameters [1]. Their work focuses on extracting and using disk parameters for real-time prediction of disk service times. To make the disk more predictable, its cache is disabled. The extraction is validated against a real disk.

As with previous work, DIXtrac determines most disk characteristics by measuring the per-request service times for specific test vectors of READ and WRITE commands. While DIXtrac uses some of the techniques outlined in previous work [18], it differs in several ways. First and foremost, the entire parameter extraction process is automated, which requires a variety of changes. Second, DIXtrac includes a simple expert system for interactively discovering a disk's layout and geometry. Third, DIXtrac's extraction techniques for cache parameters account for advances in current disk drives.

3 Characterizing Disk Drives

To completely characterize a disk drive, one must describe the disk's geometry and layout, mechanical timings, cache parameters and behavior, and all command processing overheads. Thus, the characterization of a disk consists of a list of performance-critical parameters and their values. Naturally, such a characterization makes implicit assumptions about the general functionality of a disk. For example, DIXtrac assumes that data are stored in fixed-size sectors laid out in concentric circles on rotating media.

To reliably determine most parameters, one needs a detailed disk map that identifies the physical location of each logical block number (LBN) exposed by the disk interface. Constructing this disk map requires some mechanism for determining the physical locations of specific LBNs. Using this disk map, appropriate test vectors consisting of READ and WRITE commands can be sent to the disk to extract various parameters. For many parameters, such as mechanical delays, test vectors must circumvent the cache. If the structure and behavior of the cache is known, the actual test vector can be preceded with requests that set the cache such that the test vector requests will access the media. While it is possible to devise such test vectors, it is more convenient and more efficient if the cache can be turned off.

Therefore, to accurately characterize a disk drive, there exists a set of requirements that the disk

interface must meet. First, it must be possible to determine disk's geometry either experimentally or from manufacturer's data. Second, it must be possible to read and write specific LBNs (or specific physical locations). Also, while it is not strictly necessary, it is very useful to be able to temporarily turn off the cache. With just these capabilities, DIXtrac can determine the 100+ performance-critical parameters expected by the proven DiskSim simulator.

DIXtrac currently works for SCSI disks [2], which fulfill the three listed requirements. First, the Translate option of the SEND DIAGNOSTIC and RECEIVE DIAGNOSTIC commands translates a given LBN to its physical address on the disk, given as a <*cylinder,head,sector*> tuple. SCSI also provides the READ DEFECT LIST command, which gives the physical locations of all defective sectors. With these two commands, DIXtrac can create a complete and accurate disk map. Second, the SCSI READ and WRITE commands take a starting LBN and a number of consecutive blocks to be read or written, respectively. Third, the cache can usually be enabled and disabled by changing the Cache Mode Page with the SCSI MODE SELECT command. The validation results in Section 6 show that these are sufficient for DIXtrac.

4 DIXtrac Characterization Algorithms

DIXtrac's disk characterization process can be divided into four logical steps. First, complete layout information is extracted and a disk map is created. The information in the disk map is necessary for the remaining steps, which involve issuing sequences of commands to specific physical disk locations. Second, mechanical parameters such as seek times, rotational speed, head switch overheads, and write settling times are extracted. Third, the cache management policies are determined. Fourth, command processing and block transfer overheads are measured. Since these overheads rely on information from all three of the prior steps, they must be extracted last. The remainder of this section details the algorithms used for each of these steps.

4.1 Layout Extraction

Generally, in a modern disk drive, *sectors* of host data are physically organized as concentric circles (called *tracks*) on each usable surface of the drive's stack of platters. There is a distinct read/write head for each surface, and the set of tracks equidistant from the disk center is called a *cylinder*. Different disks have different numbers of surfaces, different numbers of tracks per surface, and different numbers of sectors per track. Further, the number of sectors per track often varies from

one group (or *zone*) of cylinders to another, exploiting the larger circumferences of the cylinders farther from the center.

In addition to differences in physical storage configurations, the algorithms used for mapping the logical block numbers (LBNs) exposed by the SCSI interface to physical sectors of magnetic media vary from disk model to disk model. A common approach places LBNs sequentially around the topmost and outermost track, then around the next track of the same cylinder, and so on until the outermost cylinder is full. The process repeats on the second outermost cylinder and so on until the locations of all LBNs have been specified. This basic approach is made more complex by the many different schemes for spare space reservation and the mapping changes (e.g., reallocation) that compensate for defective media regions. Additionally, the firmware of some disks reserves part of the storage space for its own internal use.

DIXtrac's approach to disk geometry and LBN layout extraction experimentally characterizes a given disk by comparing observations to known layout characteristics. To do this, it requires two things of the disk interface: an explicit mechanism for discovering which physical locations are defective, and an explicit mechanism for translating a given LBN to its physical cylinder, surface and sector (relative to other sectors on the same track). As described in Section 3, SCSI disk drives provide these interfaces.

DIXtrac accomplishes layout extraction in several steps, which progressively build on knowledge gained in earlier steps. First, it discovers the basic physical geometry characteristics (e.g., numbers of LBNs, cylinders and surfaces) by translating random and targeted LBNs. Second, it finds out where any media defects are located. Third, explicitly avoiding defective regions, it figures out the sparing scheme (e.g., the allocation of spare sectors) used and the locations of any space reserved by the firmware. Fourth, it determines the boundaries and number of sectors per track for each zone. Fifth, the remapping mechanism used for each defective sector is identified.[1]

Steps 3–5 all exploit the regularity of geometry and layout characteristics to efficiently zero-in on the parameter values, rather than translating every LBN. DIXtrac identifies the spare space reservation scheme by determining the answers to a number of questions, including "Does each track in a cylinder have the same number of sectors?", "Does one cylinder within a set have too few sectors than can be explained by defects?", and "Does the last cylinder of a zone have too few

[1]Remapping approaches used in modern disks include slipping, wherein the LBN-to-physical location map is modified to simply skip the defective sector, and remapping, wherein the LBN that would be located at the given sector is instead located elsewhere but other mappings are unchanged. Most disks will convert to slipping whenever they are formatted via the SCSI FORMAT command.

sectors?". By combining these answers, DIXtrac decides which known scheme the disk uses; so far, we have observed eight different approaches: spare tracks per zone, spare sectors per track, spare sectors per cylinder, spare sectors per group of cylinders, spare sectors per zone, spare sectors at the end of the disk, and combinations of spare sectors per cylinder (or group of cylinders). The zone information is determined by simply counting the sectors on tracks as appropriate. The remapping scheme used for each defect is determined by back-translating the LBN that should be mapped to it (if any) and then determining to where it has been moved.

DIXtrac uses built-in expertise to discover disks' algorithms for mapping data on the disk in order to make the characterization efficient in terms of both space and time. An alternate approach would be to simply translate each LBN and maintain a complete array of these mappings. However, this is much more expensive, generally requiring over 1000X the time and 300X the result space. The price paid for this efficiency is that DIXtrac successfully characterizes only those geometries and LBN layouts that are within its knowledge base. While this knowledge base is broad and growing, it is not infinite and cannot anticipate all future directions taken by industry.

In order to avoid an extra rotation when accessing consecutive blocks across a track or cylinder boundary, disks implement track and cylinder skew. The skew must be sufficiently large to give enough time for the head switch or seek. With the cache disabled, the track and cylinder skew for each zone can be determined by issuing two WRITE commands to two consecutive blocks located on two different tracks or cylinders. The response time of the second request is measured and the value of track or cylinder skew is obtained as

$$\text{Skew} = \frac{T_{Write_2} * \text{sectors per track}}{T_{one\ revolution}}$$

4.2 Disk Mechanics Parameters

DIXtrac extracts mechanical timings with techniques detailed previously in the literature [18]. These techniques are described here for completeness.

The central technique for extraction of these parameters is to measure the minimum time between two request completions, called the *MTBRC*. As described in [18], $MTBRC(X,Y)$ denotes the minimum time between the completions of requests of type X and Y. Finding the minimum time is an iterative process in which the inter-request distance is varied until the minimal time is observed, effectively eliminating rotational latency for request Y. An *MTBRC* value is a sum of several discrete service time components, and the individual components can be isolated via algebraic manipulations as described below.

Head Switch Time. To access sectors on a different surface, a disk must switch on the appropriate read/write head. The time for head switch includes changing the data path and re-tracking (i.e., adjusting the position of the head to account for inter-surface variations). The head switch time can be computed from two *MTBRC*s:

$$\begin{aligned}\text{MTBRC}_1 &= \text{Host Delay}_1 + \text{Command} + \text{Media Xfer} + \text{Bus Xfer} + \text{Completion}\\ \text{MTBRC}_2 &= \text{Host Delay}_2 + \text{Command} + \text{Head Switch} + \text{Media Xfer} + \text{Bus Xfer} + \text{Completion}\end{aligned}$$

to get

$$\text{Head Switch} = (\text{MTBRC}_2 - \text{Host Delay}_2) - (\text{MTBRC}_1 - \text{Host Delay}_1)$$

This algorithm measures the *effective* head switch (i.e., the time not overlapped with command processing or data transfer). Therefore the value may not be physically exact, but is appropriate for disk models and schedulers [18].

Seek Times. To access data blocks, disk drives mechanically move a set of arms equipped with read and write heads. Before accessing blocks on a specific track, the arm and its heads have to be positioned over that track. This arm movement is called a seek. To extract seek times, DIXtrac uses the SEEK command that, given a logical block number, positions the arm over the track with that block. Extracted seek time for distance d consists of measuring 5 sets of 10 inward and 10 outward seeks from a randomly chosen cylinder. Seek times are measured for seeks of every distance between 1 and 10 cylinders, every 2nd distance up to 20 cylinders, every 5th distance up to 50 cylinders, every 10th distance up to 100 cylinders, every 25th distance up to 500 cylinders, and every 100th seek distance beyond 500 cylinders. Seek times can also be extracted using MTBRCs, though using SEEK is much faster. For disks that do not implement SEEK (although all of the disks we used did), DIXtrac measures $MTBRC$(1-sector write, 1-sector read) and $MTBRC$(1-sector write, 1-sector read incurring k-cylinder seek). The difference between these two values represents the seek time for k cylinders.

Write Settle Time. Before starting to write data to the media after a seek or head switch, most disks allow extra time for finer position verification to ensure that the write head is very close to the center of the track. The write settle time can be computed from head switch and a pair of $MTBRC$s: $MTBRC_1$(one-sector-write, one-sector-write on the same track) and $MTBRC_2$(one-sector-write, one-sector-write on a different track of the same cylinder). The two $MTBRC$ values are the sum of

head switch and write settle. Therefore, the head switch is subtracted from the measured time to obtain *effective* write settle time. As with head switch overhead, the value indicates settling time not overlapped with any other activity. If the extracted value is negative, it indicates that write settling completely overlaps with some other activity (e.g., command processing, bus transfer of data, etc.)

Rotational Speed. DIXtrac measures rotation speed via $MTBRC$(1-sector-write, same-sector-write). In addition, the nominal specification value can generally be obtained from the Geometry Mode Page using the MODE SENSE command.

4.3 Cache Parameters

Current SCSI disk drive controllers typically contain 512 KB to 2 MB of RAM. This memory is used for various purposes: as working memory for firmware, as a speed-matching buffer, as a prefetch buffer, as a read cache, or as a write cache. The prefetch buffer and read/write cache space is typically divided into a set of circular buffers, called segments. The cache management policies regarding prefetch, write-back, etc. vary widely among disk models.

The extraction of each cache parameter follows the same approach: construct a hypothesis and then validate or disprove it. Testing most hypotheses consists of three steps. First, the cache is polluted by issuing several random requests of variable length, each from a different cylinder. The number of random requests should be larger that the number of segments in the cache. This pollution of the cache tries to minimize the effects of adaptive algorithms and to ensure that the first request in the hypothesis test will propagate to the media. Second, a series of cache setup requests are issued to put the cache into a desired state. Finally, the hypothesis is tested by issuing one or more requests and determining whether or not they hit in the cache. In its cache characterization algorithms, DIXtrac assumes that a READ or WRITE cache hit takes at most 1/4 of the full revolution time. This value has been empirically determined to provide fairly robust hit/miss categorizations, though it is not perfect (e.g., it is possible to service a miss in less than 1/4 of a revolution). Thus, the extraction algorithms are designed to explicitly or statistically avoid the short media access times that cause miscategorization.

4.3.1 Basic Parameters

DIXtrac starts cache characterization by extracting four basic parameters. Given the results of these basic tests, the other cache parameters can be effectively extracted. The four basic hypotheses are:

Cache discards block after transferring it to the host. Do a one-block READ followed immediately by READ of the same block. If the second READ command takes more than 1/4 of a revolution to complete, then it is a miss and the cache does not keep the block in cache after transferring it to the host.

Disk prefetches data to the buffer after reading one block. Issue a READ to block n immediately followed by a READ to block $n+1$. If the second READ command completion time is more than 1/4 of a revolution, then it is a cache miss and the disk does not prefetch data after read. This test assumes that the bus transfer and host processing time will cause enough delay to incur a "miss" and a subsequent one rotation delay on reading block $n+1$ if the disk were to access the media.

WRITEs are cached. Issue a READ to block n immediately followed by a WRITE to the same block. If the WRITE completion time is more than 1/4 of a revolution, then the disk does not cache WRITEs.

READs hit on data placed in cache by WRITEs. Issue a WRITE to block n followed by a READ of the same block. If the READ completion time is more that 1/4 of a revolution, then the READ is a cache miss and READ hits are not possible on written data stored in cache.

4.3.2 Cache Segments

Number of read segments. Set the hypothesized number of segments to some number N (e.g. $N = 64$). First, pollute the cache as described above. Second, issue N READs to the first logical block LBN_k of N distinct cylinders, where $1 \leq k \leq N$. Third, probe the contents of the cache. If the disk retains the READ value in the cache after transferring it to the host, issue N READs to LBN_k; otherwise, if the disk prefetches blocks after a READ, issue N READs to $LBN_k + 1$. If any of the READs take longer than 1/4 of a revolution, assume that a cache miss occurred. Decrement N and repeat the procedure again. If all READs were cache hits, the cache uses at least N segments. In that case, increment N and repeat the algorithm.

Using binary search, the correct value of N is found. DIXtrac also determines if the disk cache uses an adaptive algorithm for allocating the number of segments by keeping track of the upper boundaries during the binary search. If an upper boundary for which there has been both a miss and a hit is encountered, then the cache uses an adaptive algorithm and the reported value may be incorrect.

DIXtrac's algorithm for determining the number of segments assumes that a new segment is

allocated for READs of sectors from distinct cylinders. It also assumes that every disk's cache either retains the READ block in cache or uses prefetching. Both assumptions have held true for all tested disks.

Number of write segments. DIXtrac counts write segments with the same basic algorithm as for read segments, simply replacing the READ commands with WRITEs. Some disks allow sharing of the same segment for READs and WRITEs. In this case, the number of write segments denotes how many segments out of the total number (which is the number of read segments) can be also used for caching WRITEs. For disks where READs and WRITEs do not share segments, simply adding the two numbers gives the total number of segments.

Size of a segment. Choose an initial value for the segment size S (e.g., the number of sectors per track in a zone). If the cache retains cached values after a transfer to the host, issue N READs of S blocks each starting at LBN_k, where N is the previously determined number of read segments. Then, probe the cache by issuing N one-block READs at LBN_k. If there were no misses, increment S and repeat the algorithm. If the cache discards the cached value after a READ, issue one-block READs to LBN_k, waiting sufficiently long for each prefetch to finish (e.g., several revolutions). Then, determine if there are hits on $LBN_k + 1$. As before, binary search is used to find the segment size S and to detect possible adaptive behavior. This algorithm assumes that prefetching can fill the entire segment. If this is not the case, the segment size may be underestimated.

4.3.3 Prefetching

Number of prefetched sectors. Issue one-block READ at LBN_1 which is the logical beginning of the cylinder. After a sufficiently long period of time (i.e., 4 revolutions), probe the cache by issuing a one-block READ to $LBN_1 + P$ where P is the hypothesized prefetch size. By selecting appropriate LBN_1 values, DIXtrac can determine the maximum prefetch size and whether the disk prefetches past track and cylinder boundaries.

Track prefetching algorithm. Some disks implement a prefetch algorithm that automatically prefetches all of track $n + 1$ only after READ requests fetch data from track $n - 1$ and n on the same cylinder [18]. To test for this behavior, first "pollute" the cache and then issue entire-track READs to track $n - 1$ and n. Then, wait at least one revolution and issue a one-block READ to a block on track $n + 1$. If there was a cache hit and the previous prefetch size indicated 0, then the disk implements this track-based prefetch algorithm.

4.3.4 Read/Write-on-Arrival

To minimize the media access time, some disks can access sectors as they pass under the head rather than in strictly ascending order. This is known as zero-latency read/write or read/write-on-arrival. To test for write-on-arrival, issue a one-block WRITE at the beginning of the track followed by an entire-track WRITE starting at the same block. If the completion time is approximately two revolutions, the disk does not implement write-on-arrival, because it takes one revolution to position to the original block and another revolution to write the data. If the time is much less, the disk implements write-on-arrival. Read-on-arrival for requested blocks is tested the same way by using READ requests.

4.3.5 Degrees of freedom provided by the Cache Mode Page

The SCSI standard defines a Cache Mode Page that allows one to set various cache parameters. However, since the Cache Mode Page is optional, typically only a subset of the parameters on the page are changeable. To determine what parameters are changeable and to what degree, DIXtrac runs several versions of the cache parameter extractions. This information is valuable for systems that want to aggressively control disk cache behavior (e.g., see [15]). The first version observes the disk's default cache behavior. Other versions explore the effects of changing different Cache Mode Page fields, such as the minimal/maximal prefetch sizes, the number/size of cache segments, the Force-Sequential-Write bit, and the Discontinuity bit. For each, DIXtrac determines and reports whether changing the fields has the specified effect on disk cache behavior.

4.4 Command Processing Overheads

DIXtrac refines and automates the MTBRC-based scheme for determining command processing overheads described in [18]. The eight extracted command overheads are listed below with how they are extracted.

MTBRC_1(1-sector write, 1-sector read on other cylinder) =

Host Delay + **Read Miss After Write** + seek + Media Xfer + Bus Xfer

MTBRC_2(1-sector write, 1-sector write on other cylinder) =

Host Delay + **Write Miss After Write** + seek + Media Xfer + Bus Xfer

MTBRC_3(1-sector read, 1-sector write on other cylinder) =

Host Delay + **Write Miss After Read** + seek + Bus Xfer + Media Xfer

MTBRC_4(1-sector read, 1-sector read miss on other cylinder) =

Host Delay + **Read Miss After Read** + seek + Media Xfer + Bus Xfer

Time_5(1-sector read hit after a 1-sector read) =

Host Delay + **Read Hit After Read** + Bus Xfer

Time_6(1-sector read hit after a 1-sector write) =

Host Delay + Bus Xfer + **Read Hit After Write**

Time_7(1-sector write hit after a 1-sector read) =

Host Delay + **Write Hit After Read** + Bus Xfer

Time_8(1-sector write hit after a 1-sector write) =

Host Delay + **Write Hit After Write** + Bus Xfer

Media transfer for one block is computed by dividing the rotational speed by the relevant number of sectors per track. Bus transfer is obtained by comparing completion times for two different-sized READ requests that are served from the disk cache. The difference in the times is the additional bus transfer time for the larger request.

When determining the MTBRC values for cache miss overheads, four different seek distances are used and appropriate seek times are subtracted. The MTBRC values are averaged to determine the overhead. Including a seek in these MTBRC measurements captures the effective overhead values given overlapping of command processing and mechanical positioning activities. Four different cylinder distances are used to account for disks which have significantly lower seek times for distance of 1 or two cylinders.

Compared to the human-assisted methodology suggested in [18], each overhead extraction is independent of the others, obviating the need for fragile matrix solutions. Also, cache hit times are measured directly rather than with MTBRC, avoiding problems of uncooperative cache algorithms (e.g., cached writes are cleared in background unless the next request arrives).

5 DIXtrac Implementation

DIXtrac runs as a regular application on the Linux 2.2 operating system. Raw SCSI commands are passed to the disk under test via the Linux raw SCSI device driver (`/dev/sg`). Each such SCSI

command is composed in a buffer and sent to the disk via a `write` system call to `/dev/sg`. The results of a command are obtained via a `read` system call to `/dev/sg`. The `read` call blocks until the disk completes the command.

DIXtrac extracts parameters in the following steps. First, it initializes the drive. Second, it performs the 4 steps of the extraction process described in Section 4. Third, it writes out parameter files and cleans up.

The initialization step first sets the drive to conform to SCSI version 2 definition via the CHANGE DEFINITION command, allowing the remainder of the extraction to use this common command set. Next, it issues 50 random read and write requests which serve to "warm up" the disk. Some drives have request times which are much longer for the first few requests after they have been sitting idle for some time. This behavior is due to several factors such as thermal re-calibration or automatic head parking.

The clean up step restores the disk to its original configuration, resetting the original SCSI version and cache settings. However, this restoration is not complete since some of the extraction steps use WRITE commands overwriting the original contents of some sectors. A possible enhancement to DIXtrac would be to save the original contents of the blocks and restore them during clean up.

The remainder of this section discusses DIXtrac's time measurement and output file formats in more detail.

5.1 Time Measurement

To measure elapsed time, DIXtrac uses the POSIX `gettimeofday` system call, which returns wall-clock time with a microsecond precision. The Linux implementation on Intel Pentium-compatible processors uses the processor's cycle counter to determine time, thus the returned time has the microsecond precision defined (but not required) by POSIX.

Before doing a `write` system call to the `/dev/sg` device, the current time is obtained via the `gettimeofday` system call. `gettimeofday` is called again after the `read` system call returns with the result of the SCSI command. The execution time of the SCSI command is the difference of those two times. The time measured via `gettimeofday` includes the overheads of the `gettimeofday` and `read/write` calls and may include a sleep time during which other processes are scheduled. The advantage of using `gettimeofday` call is that the time can be measured on an unmodified kernel.

Although it is not required for proper functioning of DIXtrac, the parameter extractions reported here were performed on a kernel with modified `sg` driver that samples the time in kernel

right before calling the low-level portion of the device driver. The measured time is returned as a part of the `sg` device structure that is passed to the `read` and `write` call. This modification eliminates the overheads of `gettimeofday` giving more precise time measurements. The times obtained via the user-level `gettimeofday` call on the unmodified kernel are, on average, 1.5% larger, with the largest deviation of 6.8%, compared to the time obtained via modified `sg` driver.

However, even the time obtained from the modified `sg` driver includes the PC bus, host adapter, and SCSI bus overheads. Bus and device driver overheads could be isolated by measuring time using a logical analyzer attached to the SCSI bus. DIXtrac assumes that, on average all SCSI commands incur the same bus and driver overheads. To eliminate bus contention issues, DIXtrac should extract data from a disk on a dedicated SCSI bus with no other device attached. The effects of other devices on the PC internal bus are minimized by performing extraction on an otherwise idle system.

5.2 Output File Formats

DIXtrac produces several output files whose names are created by appending an appropriate suffix to one of the command-line arguments, the *model-name*. To support use of the extracted data in DiskSim [3], DIXtrac creates a specification file *model-name*`.diskspecs` in the appropriate format. It also creates a configuration file *model-name*`.parv` and a seek time file *model-name*`.seek`. These three files can be used directly in the DiskSim simulator. Section 6 describes our use of DiskSim to validate DIXtrac's extraction algorithms.

6 Results and Performance

DIXtrac has been fully tested on 8 disk models: IBM Ultrastar 18ES, Hewlett-Packard C2247, Quantum Viking, Quantum Atlas III, Quantum Atlas 10K, Seagate Barracuda, Seagate Cheetah, and Seagate Hawk. This section evaluates DIXtrac in terms of extraction times and characterization accuracies, focusing on the four most recent of the tested disk drives: Ultrastar, Atlas III, Atlas 10K, and Cheetah.

6.1 Extraction Times

Table 1 summarizes the extraction times. The times are broken down to show how long each extraction step takes. With the exception of the IBM Ultrastar 18ES, an entire characterization takes

Vendor	IBM	Quantum		Seagate
Disk Model	Ultrastar 18ES	Atlas III	Atlas 10K	Cheetah
Model Number	DNES309170W	TD9100W	TN09100W	ST34501N
Capacity	9.1 GB	9.1 GB	9.1 GB	4.5 GB
Task	*Time (seconds)*			
Layout extraction	164.7 (10.6)	20.9 (0.8)	50.1 (3.9)	47.6 (0.4)
Complete seek curve	45.2 (0.1)	43.5 (0.2)	33.3 (0.3)	67.3 (0.1)
Other mechanical overheads	35.8 (1.3)	21.3 (2.5)	18.6 (1.5)	16.6 (1.4)
Cache parameters	25.6 (0.6)	8.1 (0.4)	12.6 (0.3)	12.3 (1.8)
Command process overheads	64.3 (2.5)	43.5 (1.5)	12.7 (0.9)	23 (2.3)
Totals	**335.6 (9.2)**	**137.4 (3.1)**	**127.4 (4)**	**166.8 (3.4)**

Table 1: Break down of extraction times for tested disks. The times are mean values of five extractions. The values in parentheses are standard deviations. "Other mechanical overheads" includes the extraction of head switch, write settle, and rotation speed.

less than three minutes. Extraction times could be reduced further, at the expense of accuracy, by using fewer repetitions for the timing extractions (e.g., seek, mechanical, and command processing overheads).

The extraction time for the IBM Ultrastar 18ES is longer because of the layout extraction step. The layout of this disk includes periodic unused cylinders, which causes DIXtrac to create dummy zones and repeat time-consuming, per-zone extractions (e.g., sectors per track, track skew, etc.). Extraction for this layout could certainly be optimized, but we are pleased that it worked at all given its unexpected behavior.

Table 2 shows the number of address translations required by DIXtrac to characterize each disk. Note that the number of translations does not depend directly on the capacity of the disk. Instead, it depends mainly on the sparing scheme, the number of zones, and the number of defects. More translations are performed for disks with more defects, because slipped and relocated blocks obtained from the defect list are verified. Comparing the number of blocks to the number of translations provides a metric of efficiency for DIXtrac's layout discovery algorithms. Given the time required for each translation, this efficiency is important.

The disk maps obtained by the extraction process have been verified for all eight models (24 actual disks) by doing address translation of every logical block and comparing it to the disk map information. The run time of such verification ranges from almost 5 hours (Quantum Atlas 10K) to 21 hours (Seagate Barracuda). In addition to validating the extraction process, these experiments highlight the importance of translation-count-efficiency when extracting the disk layout map.

Vendor	IBM	Quantum		Seagate
Disk Model	Ultrastar 18ES	Atlas III	Atlas 10K	Cheetah
Capacity (blocks)	17916239	17783250	17783248	8887200
Defects	123	56	64	21
One Address Translation	2.41 ms	1.66 ms	0.86 ms	7.32 ms
Translations	36911 (39)	7081 (11)	26437 (79)	5245 (43)

Table 2: Address translation characteristics for tested disks. The number of translations is the average of five runs. The values in parantheses are standard deviations.

6.2 Validation of Extracted Values

We evaluate the accuracy of parameters extracted by DIXtrac by using DiskSim. After extracting parameters from each disk drive, a synthetic trace was generated, and the response time of each request in the trace was measured on the real disk. The trace run and the extracted parameter file were then fed to DiskSim to produce simulated per-request response times. The real and simulated response times are then compared.

Two synthetic workloads were used to test the extracted parameters. The first synthetic workload was executed on each disk with both read and write caches turned off. This workload tests everything except the cache parameters. It consists of 5000 independent requests with 2/3 READs and 1/3 WRITEs. The requests are uniformly distributed across the entire disk drive. The size of the requests is between 2 and 12 KB with a mean size of 8 KB. The inter-arrival time is uniformly distributed between 0 and 72 ms.

The second workload focuses on the cache behavior and was executed with the disk's default caching policies. This trace consists of 5000 requests (2/3 READs and 1/3 WRITEs) with a mix of 20% sequential requests, 30% local (within 500 LBNs) requests, and 50% uniformly distributed requests. The size of the requests is between 2 and 12 KB with a mean size of 8 KB. The inter-arrival time is uniformly distributed between 0 and 72 ms.

For each disk, five extractions were performed to create five sets of disk parameters. For each set of parameters, five mixed traces and five random traces were generated and run on the real disk and on the DIXtrac-configured simulated disk. So, for each disk, 50 validation experiments were run. The simulated and measured response times for the tested disks are shown in Figure 1. Each curve is a cumulative distribution of all collected response times for the 25 runs, consisting of 125000 data points.

The difference between the response times of the real disk and the DIXtrac-configured simulator

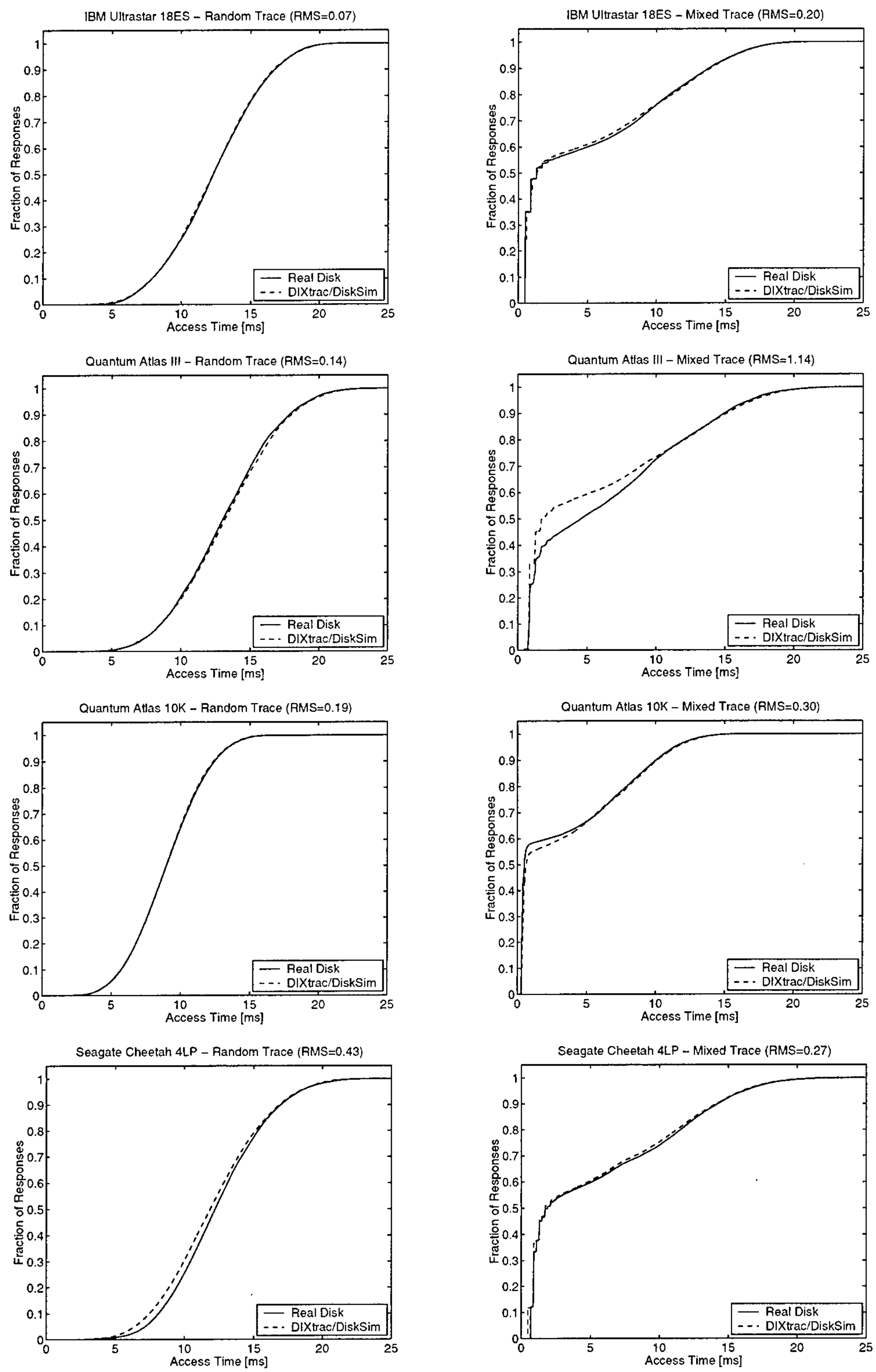

Figure 1: Measured and simulated response time cumulative distribution for IBM Ultrastar 18ES, Quantum Atlas III, Atlas 10K, and Seagate Cheetah 4LP disks.

Vendor	IBM		Quantum				Seagate	
Disk Model	Ultrastar 18ES		Atlas III		Atlas 10K		Cheetah	
Trace Type	mixed	random	mixed	random	mixed	random	mixed	random
$RMS_{Overall}$ (ms)	**0.20**	**0.07**	**1.14**	**0.14**	**0.30**	**0.19**	**0.27**	**0.43**
% $\overline{T}_{Mean}$	3.9%	0.6%	18.1%	1.1%	8.6%	2.2%	4.9%	3.5%
RMS_{Min} (ms)	0.17	0.06	1.10	0.12	0.27	0.05	0.15	0.30
% $\overline{T}_{Mean}$	3.3%	0.5%	17.4%	0.9%	7.9%	0.6%	2.7%	2.4%
RMS_{Max} (ms)	0.27	0.14	1.18	0.22	0.34	0.30	0.37	0.55
% $\overline{T}_{Mean}$	5.3%	1.1%	18.7%	1.7%	9.8%	3.4%	6.7%	4.4%

Table 3: Demetit figures (RMS) for tested disks. $RMS_{Overall}$ is the overall demerit figure for all trace runs combined. RMS_{Min} and RMS_{Max} are the minimal and maximal values of a particular run out of the 25 runs. The % $\overline{T}_{Mean}$ value is the percent difference of the respective RMS from the mean real disk response time.

can be quantified by a demerit figure [10], which is the root mean square distance between the two curves. The demerit figure, here referred to as the RMS, for each graph is given in Table 3. Most of these accuracies compare favorably with the most accurate disk simulation models reported in the literature [10, 18].

However, several suboptimal values merit discussion. For the Seagate Cheetah, the simulated disk with DIXtrac-extracted parameters services requests faster than the real disk. This difference is due to smaller effective values of READ command processing overheads. Manually increasing these values by 0.35 ms results in a closer match to the real disk with RMS at 0.17 ms and 0.16 ms for the mixed and random trace respectively. We have not yet been able to determine why DIXtrac underestimates this command overhead.

The differences in the mixed trace runs for the two Quantum disks are due to shortcomings of DiskSim. DIXtrac correctly determines that the disks use adaptive cache behavior. However, because DiskSim does not model such caches, DIXtrac configures it with average (disk-specific) values for the number and size of segments. The results show that the actual Atlas 10K disk has more cache hits than the DiskSim model configured with 10 segments of 356 blocks. Interestingly, the adaptive cache behavior of the real Atlas III disk is worse than the behavior of the simulated disk configured with 6 segments and 256 blocks per segment. Manually lowering the value of blocks per segment to 65, while keeping all other parameters the same, gives the best approximation of the real disk behavior.

These empirical validation results provide significant confidence in the accuracy of the parameters extracted by DIXtrac. For additional confidence, extracted data were compared directly with

the specifications given in manufacturer's technical manuals [12, 11, 4, 7, 8, 6], wherever possible. In all such cases, the extracted values match the data in the documentation.

7 Summary

DIXtrac is a program that automatically characterizes the performance of modern disk drives. This paper describes and validates DIXtrac's algorithms, which extract accurate values for over 100 performance-critical parameters. By configuring a detailed disk simulator with the parameters extracted by DIXtrac, accurate disk simulation models are demonstrated; the resulting accuracies match those of the most accurate disk simulators reported in the literature. To date, DIXtrac has been successfully used on over 20 disk drives, consisting of eight models of disk drives from four different manufacturers.

With DIXtrac as a tool, we are providing a database of validated disk characterizations on the Web (`http://www.ece.cmu.edu/~schindjr/dixtrac`). DIXtrac output files, including parameter files usable by the freely-available DiskSim simulator.

Acknowledgements

This work was supported in part by the National Science Foundation under grant #ECD-8907068. The authors are indebted to generous contributions from the member companies of the Parallel Data Consortium, including: Hewlett–Packard Laboratories, Intel, Quantum, Seagate Technology, 3Com Corporation, CLARiiON Array Development, LSI Logic, EMC Corporation, Hitachi, Veritas, Procom Technology, MTI Technology, Infineon Technologies, and Novell, Inc.

References

[1] M. Aboutabl, A. Agrawala, and J. Decotignie. Temporally determinate disk access: An experimental approach. Technical Report CS-TR-3752, Dept. of Computer Science, University of Maryland, College Park, 1997.

[2] *Small Computer System Interface-2*, draft revision 10k edition, March 1993.

[3] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment. Technical Report CSE-TR-358-98, Dept. of Electrical Engineering and Computer Science, Univ. of Michigan, February 1998.

[4] Hewlett Packard Company. *HP C2244/45/46/47 3.5-inch SCSI-3 Disk Drive Technical Reference Manual*, September 1992.

[5] D. Kotz, S. B. Toh, and S. Radhakishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report PCS-TR94-220, Dept. of Computer Science, Darthmouth College, July 1994.

[6] Quantum Corporation. *Quantum Viking 2.27/4.55 GB S Product Manual*, April 1997.

[7] Quantum Corporation. *Quantum Atlas III SCSI Hard Disk Drives: Ultra SE SCSI-3 Version Product Manual*, April 1998.

[8] Quantum Corporation. *Quantum Atlas 10K 9.1/18.2/36.4 GB Ultra 160/m S Product Manual III SCSI Hard Disk Drives: Ultra SE SCSI-3 Version*, August 1999.

[9] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS apporach. *IEEE Journal of Parallel and Distributed Technology*, pages 34–43, Winter 1995.

[10] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, March 1994.

[11] Seagate Technology. *Cheetah 4LP Family: ST34501N/W/WC/DC Product Manual*, April 1996.

[12] Seagate Technology. *Barracuda 4LP Family: ST32171N/W/WC/DC Product Manual*, February 1998.

[13] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Winter USENIX Conference*, pages 313–323, 1990.

[14] P. J. Shenoy and H. M. Vin. Cello: A disk scheduling framework for next generation operating systems. In *Proc. of ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 44–55, 1998.

[15] E. Shriver, C. Small, and K. Smith. Why does file system prefetching work? In *USENIX Technical Conference*, pages 71–83, 1999.

[16] N. Talagala, R. Arpaci-Dusseau, and D. Patterson. Microbenchmark-based extraction of local and global disk characteristics. `http://www.cs.berkeley.edu/~nisha/bench.html`, 1999.

[17] A. Varma and Q. Jacobson. Destage algorithms for disk arrays with non-volatile caches. In *IEEE International Symposium on Computer Architecture*, pages 83–95, 1995.

[18] B. L. Worthington, G. R. Ganger, and Y. N. Patt. On-line extraction of SCSI disk drive parameters. In *Proc. of ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 146–156, 1995.